GPS Solut (2010) 14:293–299
DOI 10.1007/s10291-009-0149-9

GPS TOOL BOX

GNSS data management and processing with the GPSTk

Dagoberto Salazar · Manuel Hernandez-Pajares · Jose M. Juan · Jaume Sanz

Published online: 12 November 2009

Abstract We organize complex problems in simple ways using a GNSS data management strategy based on “GNSS Data Structures” (GDS), coupled with the open source “GPS Toolkit” (GPSTk) suite. The code resulting from using the GDS and their associated “processing paradigm” is remarkably compact and easy to follow, yielding better code maintainability. Furthermore, the data abstraction allows flexible handling of concepts beyond mere data encapsulation, including programmable general solvers. An existing GPSTk class can be modified to achieve the goal. We briefly describe the “GDS paradigm” and show how the different GNSS data processing “objects” may be combined in a flexible way to develop data processing strategies such as Precise Point Positioning (PPP) and network-based PPP that computes satellite clock offsets on-the-fly.

Keywords GPSTk · GNSS data structures · GDS · PPP · POP

The GPS Tool Box is a column dedicated to highlighting algorithms and source code utilized by GPS engineers and scientists. If you have an interesting program or software package you would like to share with our readers, please pass it along; e-mail it to us at gpstoolbox@ngs.noaa.gov. To comment on any of the source code discussed here, or to download source code, visit our website at http://www.ngs.noaa.gov/gps-toolbox. This column is edited by Stephen Hilla, National Geodetic Survey, NOAA, Silver Spring, Maryland, and Mike Craymer, Geodetic Survey Division, Natural Resources Canada, Ottawa, Ontario, Canada

D. Salazar (✉) · M. Hernandez-Pajares · J. M. Juan · J. Sanz
gAGE/UPC. Universitat Politécnica de Catalunya,
Barcelona, Spain
e-mail: Dagoberto.Jose.Salazar@upc.edu
URL: http://www.gage.es

Introduction

The “GPS Toolkit” (GPSTk) project (Tolman et al. 2004; Harris and Mach 2007) is an open source project sponsored by the Applied Research Laboratories of the University of Texas (ARL:UT), having several collaborators around the world. It aims to provide a GNSS computing suite to the satellite navigation community, consisting of a core library, accessory libraries, and some applications. The GPSTk functionality has been continuously improving (Renfro et al. 2005; Harris et al. 2006, 2007), and its GNU Lesser General Public License (LGPL) allows users the freedom to develop both free and commercial software.

Shortly after starting to develop carrier phase-based capabilities for the GPSTk, some project developers started to face, with increasing frequency, several data management issues that were difficult to deal with using just vectors and matrices. The present paper introduces a novel approach to GNSS data processing software development, based on a hierarchy of data structures which cope with data management issues in a consistent way. This is the origin of the “*GNSS Data Structures*” (GDS) and the associated “*GDS Processing Paradigm*”.

GNSS data structures

In order to solve the GNSS data management problem in a flexible, consistent, and comprehensive way, the “*GNSS Data Structures*” were developed and added to the **procframe** auxiliary library of the GPSTk. First introduced in Harris et al. 2007, the GDS and their associated “*GDS Processing Paradigm*” have been continuously evolving and improving since their inception (Salazar et al. 2008, 2009).

The GDS hold several kinds of GNSS-related data, indexed by **station** (*SourceID*), **epoch** (*DayTime*), **satellite** (*SatID*), and **type** (*TypeID*). In this way, both the data and corresponding "*metadata*" (data relationships) are preserved, and data management issues are properly addressed. The indexing is done automatically (i.e., without researcher explicit intervention) because classes conforming to the "*GDS Processing Paradigm*" must fulfill some requirements including proper metadata handling and data indexing. GDS take advantage of the observation that several types of GNSS-related data structures share some common characteristics, and thus they can be handled in a unified way. Please refer to GPSTk's Application Programming Interface (API) document for details: http://www.gpstk.org/doxygen.

GDS processing paradigm

Apart from the GDS themselves, a "GDS Processing Paradigm" was also developed, where the GNSS Data Structures are complemented with several associated *processing classes*. With the *GDS paradigm,* the GNSS data processing becomes like an *assembly line*, where all of the processing steps are performed sequentially. The GDS are treated like *white boxes* that "*flow*" from one "*workstation*" (processing step) to the next in this assembly line.

Thus, the GDS are always used as both the input and output of each processing step, providing an easy and straightforward way to encapsulate and process data. This paradigm allows developing clean, simple to read, and easy to use software that speeds up development and reduces errors.

The objects from these *processing classes* reach into the GDS and add, delete, and/or modify what is needed (according to their function), and leave the results in the same GDS, appropriately indexed. These processing objects are designed to use sensible defaults in their parameters, but may be tuned to suit specific needs.

The former ideas are coupled with a redefinition of C++ operator "≫", implemented in such a way that several operators may be concatenated. It allows a programming style that clearly shows how the data is *flowing* along the processing steps (resembling the "pipes" concept used in UNIX-based systems). The following two experiments show how this is done.

Experiment #1

This experiment deals with a "Precise Point Positioning" (PPP) implementation (Kouba and Heroux 2001). PPP is a complex task, and issues like phase wind-up effects; solid-, oceanic-, and polar-tides; and antenna phase centers variations must be taken into account. The International GNSS Service (IGS) precise orbits and clock files are used in PPP, but the orbits are typically provided every 900s while the observations are often collected every 30 s. Therefore, time management issues arise. The GPSTk provides some accessory classes that ease these complex issues. The implementation details for these can be found in example8.cpp, example9.cpp, and example10.cpp on the GPSTK web site: http://www.gpstk.org/doxygen/examples.html.

The code of the core PPP processing line follows:

```
    gpsData >> reqObs >> linear1 >> csLI
            >> csMW >> markArc >> decimateData
            >> basicModel >> eclipsedSV >> grDelay
            >> svPcenter >> corr >> windup
            >> cTropo >> linear2 >> pcFilter
            >> phaseAlign >> linear3 >> baseChange
            >> cDOP >> pppSolver;
```

The GDS processing data chain is a single C++ line, although in this case (for clarity sake) it spans seven physical lines. This line must be enclosed within a *while* loop to process all available epochs, and also within a *try-catch* block to manage exceptions. Table 1 provides a brief explanation about what these objects (processing steps) do and which classes they belong to.

The object *pppSolver*, belonging to the *SolverPPP* class, deserves particular mention. This object is an Extended Kalman Filter (EKF) preconfigured to solve the PPP equation system in a way consistent with Kouba and Heroux 2001: coordinates are treated as constants (static), the receiver clock is considered white noise, and the vertical wet tropospheric delay is processed as a random walk stochastic model (using the Niell mapping functions). All of these parameters are configurable.

Static PPP results

Figure 1 plots the results from this PPP processing code applied to station MADR, May 27th., 2008, using the default configuration for *SolverPPP* objects (PPP with static coordinates) and a NEU coordinate frame. The a priori position used was the one provided by the IGS in Solution Independent Exchange (SINEX) files for that epoch. These results are consistent with what it is expected from this processing strategy, showing a small residual bias in the "Up" coordinate of about 17 mm, reaching errors below 5 cm in less than 1.5 h of processing.

Figure 2 plots the 3D-positioning differences with respect to the IGS nominal position using several PPP processing tools provided by "The Precise Point

Table 1 Processing steps/objects used for precise point positioning (PPP)

Object/(Class)	Description
reqObs (RequireObservables)	Checks if required TypeID's are present
linear1 (*ComputeLinear*)	Computes linear combinations used to detect cycle slips (ionospheric, Melbourne-Wubbena)
csLI (*LICSDetector2*)	Detect cycle slips using ionospheric and Melbourne-Wubbena combinations
csMW (*MWCSDetector*)	
markArc (*SatArcMarker*)	Keeps track of satellite arcs
decimateData (*Decimate*)	If not a multiple of 900 s, then decimates data
basicModel (*BasicModel*)	Computes the basic components of a GNSS signal propagation model
eclipsedSV (*EclipsedSatFilter*)	Removes from GDS satellites in eclipse
grDelay (*GravitationalDelay*)	Computes gravitational delay effect due to changing gravity field along SV-RX ray
svPcenter (*ComputeSatPCenter*)	Computes the effect of satellite antenna phase center
corr (*CorrectObservables*)	Corrects observables from tides, receiver antenna phase center, eccentricity, etc
windup (*ComputeWindUp*)	Computes phase wind-up correction
cTropo (*ComputeTropModel*)	Models delays due to troposphere
linear2 (*ComputeLinear*)	Computes ionosphere-free combinations for code (PC) and phase (LC)
pcFilter (*SimpleFilter*)	Filters out spurious values in PC combination
phaseAlign (*PhaseCodeAlignment*)	Aligns phase with code values, preserving the integer nature of ambiguities
linear3 (*ComputeLinear*)	Computes code and phase prefilter residuals
baseChange (*XYZ2NEU*)	Prepares GDS to use a North-East-Up reference frame in *pppSolver*
cDOP (*ComputeDOP*)	Computes DOP values
pppSolver (*SolverPPP*)	Solves the equation system with an Extended Kalman Filter (EKF) configured in PPP mode

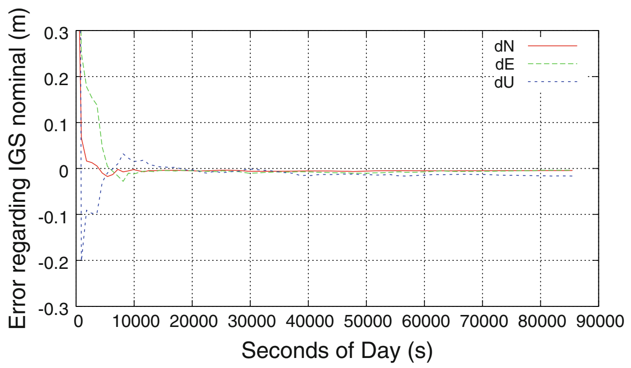

Fig. 1 GPSTk static PPP results, regarding IGS nominal, for MADR station, 2008/05/27

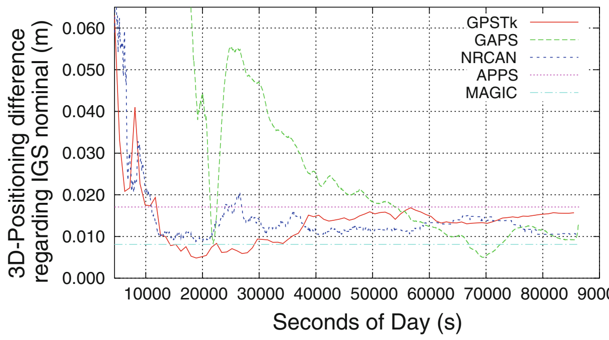

Fig. 2 3D Positioning difference regarding IGS nominal for several PPP processing tools. Static PPP results for MADR station, 2008/05/27

Positioning Software Centre" (http://gge.unb.ca/Resources/PPP). This tool receives RINEX observation files and sends them to several online PPP processing facilities such as:

- CSRS-PPP (NRCAN): http://www.geod.nrcan.gc.ca/online_data_e.php
- GPS Analysis and Positioning Software (GAPS): http://gaps.gge.unb.ca/
- Automatic Precise Positioning Service (APPS), formerly Auto-GIPSY: http://apps.gdgps.net/
- MagicGNSS (MAGIC): http://magicgnss.gmv.com/ppp

This figure confirms that our relatively simple GPSTk-based PPP code compares quite favorably both in precision and convergence time with other PPP processing tools (note that APPS and MAGIC work in static, forward–backward mode, providing only the last position solution).

Kinematic PPP results

The *pppSolver* preassigned *stochastic models* may be tuned and changed at will, given that they are objects inheriting from a general class called *StochasticModel*. This is a very important advantage of abstraction, and treating coordinates as white noise (kinematic mode) may be achieved in a very simple way:

```
1  WhiteNoiseModel newCoordinatesModel(100.0);
2  pppSolver.setCoordinatesModel(&newCoordinatesModel);
```

In line #1, a white noise stochastic model object (with a sigma of 100 m) is declared, while in line #2, the *pppSolver* object is configured to use the new model for coordinate estimation. The vertical wet tropospheric effect is still treated as a random walk process, and the receiver clock continues as another white noise process (with a higher sigma). Figure 3 presents the results, confirming the good quality of GPSTk model: the coordinates are consistently within ±10 cm of the IGS values, with a 3D-RMS of 0.047 m for the convergence phase (from 1.5 h onwards).

The services of the "The Precise Point Positioning Software Centre" were used again, this time to compute the kinematic positioning. Table 2 presents the 3D-RMS position difference with respect to the IGS SINEX position (for the convergence phase). MagicGNSS results are not shown because it provides only static solutions. GPSTk delivers the second lowest 3D-RMS.

Forward–backward PPP results

The previous results were obtained with a Kalman filter that only runs forward. However, given that PPP is done in post-processing mode, a stronger solution can be obtained

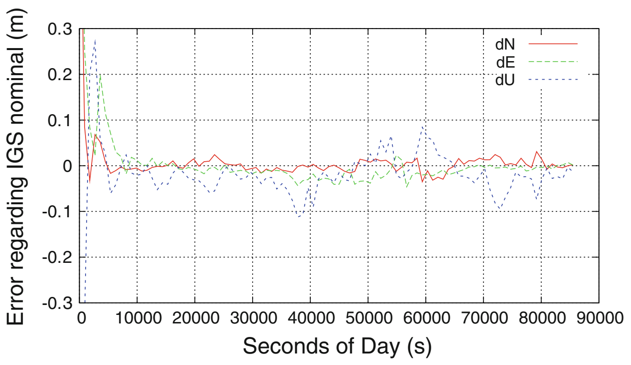

Fig. 3 Kinematic PPP results, regarding IGS nominal, for MADR station, 2008/05/27

Table 2 3D-RMS for Kinematic PPP position differences regarding IGS SINEX solution

PPP positioning tool	3D-RMS (m)
APPS	0.034
GAPS	0.067
GPSTk	0.047
NRCAN	0.073

running the filter in forward–backward mode, where ambiguity convergence achieved in a given forward run is used for the next backward run. It may be iterated at will.

An object of class *SolverPPPFB* is used for this. It encapsulates *SolverPPP* class functionality and adds a data management and storage layer to handle the whole process. From the user's point of view, the main change is to replace the *SolverPPP* object (*pppSolver*) with a new *SolverPPPFB* object (*fbpppSolver*) inside the *while* loop that reads and processes the RINEX observation file.

After the first forward processing is done (and data is internally indexed and stored), it is simply a matter of telling the *fbpppSolver* object how many forward–backward cycles we want it to "re-process". For instance:

```
fbpppSolver.ReProcess(4);
```

After that, one last forwards processing is needed to get the time-indexed solutions out of *fbpppSolver*:

```
while( fbpppSolver.LastProcess(gpsData) )
{
   cout << fbpppSolver.getSolution(TypeID::dLat) << "  ";
   cout << fbpppSolver.getSolution(TypeID::dLon) << "  ";
   cout << fbpppSolver.getSolution(TypeID::dH)   << endl;
}
```

In this case, the forward–backward processing (in static mode) is used to compute the zenith path delay estimation (zpd) for the full day. Figure 4 shows the results for APPS, MAGIC, GPSTk, and NRCAN, as well as the official, combined IGS zpd. NRCAN only provides forward estimates, and GAPS does not provide zpd estimations. Table 3 presents the RMS of the zpd differences with respect to IGS values.

Experiment #2

In this experiment, class *SolverGeneral* will be used to implement a kinematic PPP-like processing based on a network of stations, where ***satellite clock offsets will be estimated on-the-fly***. This procedure is independent of

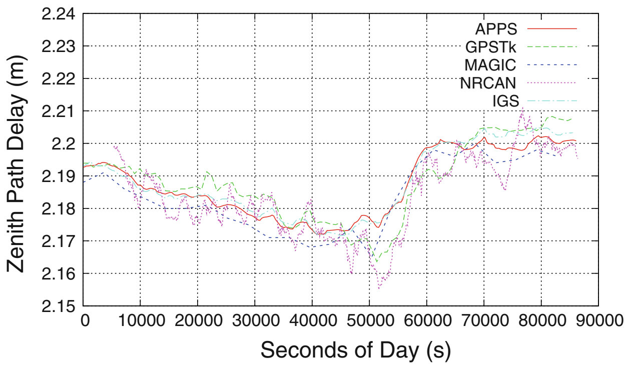

Fig. 4 Forward–backward static PPP zenith path delay results for MADR station, 2008/05/27

Table 3 RMS for zpd differences regarding IGS combined solution

PPP positioning tool	zpd RMS (m)
APPS	0.0018
GPSTk	0.0048
MAGIC	0.0052
NRCAN	0.0069

precise clock information and only needs precise orbits to work; therefore, it will be called "Precise Orbits Positioning" (POP).

The procedure starts with selecting a set of reference stations and setting one of them as ***master station***. Master's clock will be set as the reference for the network, so all the other clocks will be computed with respect to it. The other unknowns for the **master** station will be the zenith tropospheric delay and the ambiguities. Therefore, the corresponding equations for pseudorange and phase are

$$PrefitPC_0^j = tmap_0^j.ztd_0 - c\,.dt^j \tag{1}$$

$$PrefitLC_0^j = tmap_0^j.ztd_0 + Bc_0^j - c.dt^j \tag{2}$$

where $PrefitPC_0^j$ and $PrefitLC_0^j$ are, respectively, the prefilter residual (observation minus modeled effects) of ionosphere-free code and phase combination for satellite j and master station "0". Further, $tmap_0^j$ is the tropospheric mapping function, ztd_0 is the zenith tropospheric path delay, $c.dt^j$ is the relative clock delay between satellite j and master station "0" in meters, and Bc_0^j is the ionosphere-free phase ambiguity.

The other **reference** stations will have similar equations, but adding their clock offsets (with respect to master clock) as an additional unknown. Hence,

$$PrefitPC_k^j = tmap_k^j.ztd_k + c.dt_k - c.dt^j \tag{3}$$

$$PrefitLC_k^j = tmap_k^j.ztd_k + Bc_k^j + c.dt_k - c.dt^j \tag{4}$$

where $c.dt_k$ is the relative clock delay between reference station k and master (meters).Finally, the "**rover**" receiver will have an equation similar to the standard PPP process, but adding the estimation of satellite clock offsets gives

$$\begin{aligned}PrefitPC_r^j = {} & ((x_{r0} - x^j)/\rho_{r0}^j)dx + ((y_{r0} - y^j)/\rho_{r0}^j)dy \\ & + ((z_{r0} - z^j)/\rho_{r0}^j)dz + tmap_r^j.ztd_r + c.dt_r \\ & - c.dt^j\end{aligned} \tag{5}$$

$$\begin{aligned}PrefitLC_r^j = {} & ((x_{r0} - x^j)/\rho_{r0}^j)dx + ((y_{r0} - y^j)/\rho_{r0}^j)dy \\ & + ((z_{r0} - z^j)/\rho_{r0}^j)\,dz + tmap_r^j.ztd_r \\ & + Bc_r^j + c.dt_r - c.dt^j\end{aligned} \tag{6}$$

where (x_0, y_0, z_0) is the a priori receiver position, (x^j, y^j, z^j) is the position of satellite j, and parameters (dx, dy, dz) are the corrections to (x_0, y_0, z_0).

It can be seen that the connection between receivers is achieved by the simultaneous estimation of satellite clock offsets. As said, this procedure allows rover precise positioning without precise satellite clock products.

Implementation of this equation system starts with declaration and initialization of the *Variable* objects to be used, as well as their associated stochastic models:

```
1   WhiteNoiseModel coordinatesModel( 100.0 );
2   TropoRandomWalkModel tropoModel;
3   PhaseAmbiguityModel ambiModel;

4   Variable dLat( TypeID::dLat, &coordinatesModel, true, false, 100.0 );
5   Variable dLon( TypeID::dLon, &coordinatesModel, true, false, 100.0 );
6   Variable dH( TypeID::dH, &coordinatesModel, true, false, 100.0 );

7   Variable cdt( TypeID::cdt );
       cdt.setDefaultForced(true);   // Force default coefficient (1.0)

8   Variable tropo( TypeID::wetMap, &tropModel, true, false, 10.0 );

9   Variable ambi( TypeID::BLC, &ambiModel, true, true );
       ambi.setDefaultForced(true);   // Force default coefficient

10  Variable satClock( TypeID::dtSat, false, true );
       satClock.setDefaultCoefficient(-1.0); // Set default coefficient
       satClock.setDefaultForced(true);      // Force default coefficient

11  Variable prefitPC( TypeID::prefitC );
12  Variable prefitLC( TypeID::prefitL );
```

Lines #1 to #3 set the stochastic models to be used. Line #4 declares a *Variable* called *dLat*, of *TypeID* "dLat", with a white noise stochastic model (kinematic positioning). The first "true" parameter indicates that this *Variable* is "source-indexed" (i.e., it is a distinct variable for each *SourceID*), and the following "false" parameter tells that it is **not** "satellite-indexed", meaning that the same variable will be used for all visible satellites. The final numeric value (100.0) sets the initial sigma. Variables *dLon* and *dH* (lines #5, #6) follow the same pattern.

Line #7 declares *cdt*, the *Variable* representing *receiver* clock offsets. The defaults are used (white noise model, source-indexed, not satellite-indexed, big preset sigma), and it is forced to always use the value "1.0" as coefficient (by default, coefficients are looked for inside the GDS).

Declaration of variables *tropo*, *ambi* (ambiguities), and *satClock* (SV clock offsets) are similar, with the exception that ambiguities are source- **and** satellite-indexed, whereas satellite clocks are only satellite-indexed. The last couple of lines (#11, #12) declare default, dummy "variables" representing the independent terms of equations, *prefitPC* and *prefitLC*.

Once the *Variables* are properly declared and initialized, it is the turn of the *Equation* objects. First, let's declare the equations for **master** station:

```
1   Equation equPCMaster( prefitPC );
2   equPCMaster.addVariable( tropo );
3   equPCMaster.addVariable( satClock );
4   equPCMaster.header.equationSource = master;

5   Equation equLCMaster( prefitLC );
6   equLCMaster.addVariable( tropo );
7   equLCMaster.addVariable( satClock );
8   equLCMaster.addVariable( ambi );
9   equLCMaster.header.equationSource = master;
10  equLCMaster.setWeight( 10000.0 );
```

Line #1 declares the *Equation* object for pseudorange, setting the independent term type. Then, lines #2 and #3 add the variables to the equation and finally #4 sets what receiver (or data source) this equation applies to; *master* is an object of class *SourceID* holding the information corresponding to the master station. Declaration of the equation for phase is very similar, except for line #8, that adds another variable (*ambi*), and line #10 that sets the ***relative*** weight of this equation: the phase sigma is 100 times smaller, so the associated weight is 100*100 times larger.

Equations for reference stations and rover receiver are declared in the same way. However, it must be noted that reference stations form a *SourceID* **set**, instead of a single station, so they need an additional treatment. Thus, *equPCRef* and *equLCRef* are the equations for the reference stations' pseudorange and phase, respectively:

```
1   equPCRef.header.equationSource = Variable::someSources;
2   equLCRef.header.equationSource = Variable::someSources;

3   for( std::set<SourceID>::const_iterator itSet = refStationSet.begin();
         itSet != refStationSet.end();
         ++itSet )
    {
4     equPCRef.addSource2Set( (*itSet) );
5     equLCRef.addSource2Set( (*itSet) );
6   }
```

The special *SourceID* called "*Variable::someSources*" indicates that equations *equPCRef* and *equLCRef* will apply to more than one data source. Thus, it is necessary to add those data sources to each equation's internal set. The "for" loop spanning from line #3 to #6 achieves this in a general, reusable way.

Finally, the former *Equation* objects are added to an *EquationSystem*, which in turn feeds a *SolverGeneral* object:

```
1   EquationSystem system;
2   system.addEquation( equPCRover );
3   system.addEquation( equLCRover );
4   system.addEquation( equPCRef );
5   system.addEquation( equLCRef );
6   system.addEquation( equPCMaster );
7   system.addEquation( equLCMaster );

8   SolverGeneral solver( system );
```

From now on, object *solver* is a Extended Kalman Filter configured to solve the defined equation system, building its internal matrices and vectors automatically according to the incoming data. It just needs to be fed with the appropriate GDS.

The approach to this multi-station problem is to preprocess all the stations, one by one, in a way similar to Experiment #1 (PPP), but without applying the solver. The results from preprocessing are stored in an appropriate multi-epoch, multi-station GNSS data structure that automatically takes care of all indexing (structure *gnssDataMap* is used for this). Then, an epoch-worth of data is extracted each time from the *gnssDataMap* GDS and fed to *solver*, and the results are printed.

For this experiment, 5 IGS stations were used: ACOR, MADR, SCOA, SFER, and TLSE, forming a network across Iberic peninsula spanning 1023 km (SFER-TLSE). Station ACOR was set as "master", while MADR was the "rover", 392 km away from nearest reference station (SCOA).

Standard IGS products (precise orbits and satellite clocks) with a 900 s data rate were used, but the data were processed at 30 s, the rate given by the RINEX observation files. Note that in this case, the IGS satellite clocks were **not** interpolated, but **ignored**: The SV clocks used for this POP positioning were estimated on-the-fly.

Figure 5 shows the good results from this approach, presenting both the 3D-error in position (with respect to the known IGS position) of POP, and the 3D-error for the standard kinematic PPP processing (Experiment #1). The results are very similar (as expected): 3D-RMS of 0.047 m

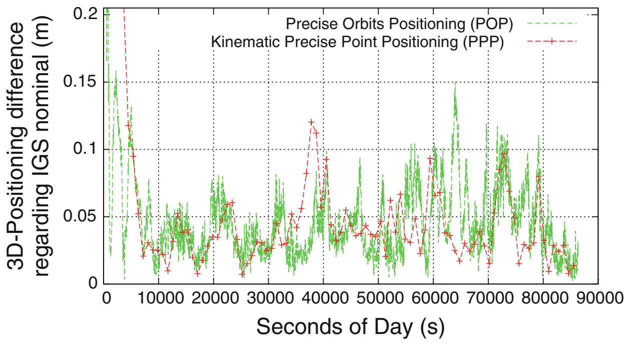

Fig. 5 3D Positioning difference regarding IGS nominal for kinematic PPP and POP processing strategies. MADR station, 2008/05/27

for the kinematic PPP case versus 3D-RMS of 0.050 m for the POP case, but POP yields a higher positioning rate.

This experiment has confirmed how the GPSTk-provided GDS, with their associated paradigm, allow one to develop code that is simple to read and maintain, but able to carry out complex GNSS data processing in an effective way.

Conclusions

This work introduced a set of extensible data structures, called "GNSS Data Structures" (GDS). The GDS, coupled with the facilities provided by the GPSTk, enable an innovative and flexible data management strategy called the "GDS paradigm", making it possible to easily develop different data processing strategies. In order to emphasize the aforementioned issues, a couple of GNSS data-processing experiments and their results were presented, including Precise Point Positioning (PPP) and Precise Orbits Positioning (POP).

The code resulting from using the GDS and their processing paradigm is remarkably clean, compact, and easy to follow; yielding better code maintainability and supporting our design goal, which is "to free researchers to focus on research". Its performance compares very favorably with other, more established, GPS processing suites. Furthermore, the GDS design is based on data abstraction, allowing a very flexible handling of concepts beyond mere data encapsulation, including programmable general solvers, among others.

References

Harris RB, Mach RG (2007) The GPSTk: an open source GPS toolkit. GPS Solut 11:145–150. doi:10.1007/s10291-006-0043-7

Harris RB, Craddock T, Conn T, Gaussiran T, Hagen E, Hughes A, Little J, Mach R, Nelsen S, Renfro B, Tolman B (2006) Open signals, open software: two years of collaborative analysis using the GPS toolkit. In: Proceedings of the 19th international technical meeting of the satellite division of the institute of navigation (ION GNSS 2006) September, Fort Worth, TX, USA

Harris RB, Conn T, Gaussiran T, Kieschnick C, Little J, Mach R, Munton D, Renfro B, Nelsen S, Tolman B, Vorce J, Salazar D (2007) The GPSTk: new features, applications and changes. In: Proceedings of the 20th international technical meeting of the satellite division of the institute of navigation (ION GNSS 2007) September, Fort Worth, TX, USA

Kouba J, Heroux P (2001) Precise point positioning using IGS orbit and clock products. GPS Solut 5:12–28. doi:10.1007/PL00012883

Renfro B, Harris RB, Tolman B, Gaussiran T, Munton D, Little J, Mach R, Nelsen S (2005) The open source GPS toolkit: a review of the first year. In: Proceedings of the 18th international technical meeting of the satellite division of the institute of navigation (ION GNSS 2005) September, Long Beach, CA, USA

Salazar D, Hernandez-Pajares M, Juan JM, Sanz J (2008) High accuracy positioning using carrier-phases with the open source GPSTk software. In: Proceedings of the 4th. ESA workshop on satellite navigation user equipment technologies (NAVITEC 2008) December, Noordwijk, The Netherlands

Salazar D, Sanz J, Hernandez-Pajares M (2009) Phase-based GNSS data processing (PPP) with the GPSTk. In: Proceedings of the 8th geomatic week, February, Barcelona. Spain

Tolman B, Harris RB, Gaussiran T, Munton D, Little J, Mach R, Nelsen S, Renfro B, Schlossberg D (2004) The GPS toolkit–open source GPS software. In: Proceedings of the 17th international meeting of the satellite division of the institute of navigation (ION GNSS 2004) September, Long Beach, CA, USA

Author Biographies

Dagoberto Salazar is an Aeronautical Engineer (IUPFAN, Venezuela, 1992). After working several years both for government and private industry, he pursued postgraduate studies in Instrumentation and Control (UCV). He is currently a PhD Candidate in Aerospace Science and Technology at the Polytechnical University of Catalonia (UPC), as well as official developer of the GPS Toolkit project (GPSTk).

Dr. Manuel Hernandez-Pajares is an associate professor of the Department of Applied Mathematics IV at the Polytechnical University of Catalonia (UPC) since 1993. He started working on GPS in 1989 for cartographic and surveying applications. Since 1995, his focus has been in the area of GNSS ionospheric determination and precise radionavigation. He was the Ionosphere WG chairman and Ionospheric product coordinator of the International GPS Service (IGS) from 2002 to 2009.

Dr. Jose M. Juan is an associate professor of the Department of Applied Physics at the Polytechnical University of Catalonia (UPC). His current research interest is in the area of GPS ionospheric tomography, GPS data processing algorithms, and radionavigation.

Dr. Jaume Sanz is an associate professor of the Department of Applied Mathematics IV at the Polytechnical University of Catalonia (UPC). His current research interest is in the area of GPS data processing Algorithms, GPS ionospheric tomography, Satellite-Based Augmentation Systems (SBAS) and Precise Radio Navigation.